1-1-1991

Multipacket Routing on Rings

Fillia Makedon
Dartmouth College

Adonios Simvonis
University of Sydney

MULTIPACKET ROUTING ON RINGS

Fillia Makedon
Adonios Simvonis

Technical Report PCS-TR91-163

Multipacket Routing on Rings

Fillia Makedon[1]
Computer Science Program
University of Texas at Dallas
P.O. Box 830688, MP 3.1
Richardson, TX75083 – 0688
makedon@utdallas.edu

Adonios Simvonis
Basser Department of Computer Science
University of Sydney
N.S.W 2006
Australia
simvonis@cs.su.oz.au

Abstract. We study multipacket routing problems. We divide the multipacket routing problem into two classes, namely, distance limited and bisection limited routing problems. Then, we concentrate on rings of processors. Having a full understanding of the multipacket routing problem on rings is essential before trying to attack the problem for the more general case of r-dimensional meshes and tori. We prove a new lower bound of $\frac{2n}{3}$ routing steps for the case of distance limited routing problems. We also give an algorithm that tightens this lower bound. For bisection limited problems, we present an algorithm that completes the routing in near optimal time.

1 Introduction

A great deal of work has been devoted to the study of the packet routing problem [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. This is because the packet routing problem is closely related to parallel computation. Through the routing of messages (packets) we are able to emulate shared memory [11]. More generally, for a parallel computer to be computationally effective, it must be able to route messages from their origin processors to their destination processors fast and with small, preferably constant size queues. These queues are created while two or more packets are waiting to cross the same communication channel.

In this paper, we consider two types of packet routing problems, namely, *distance limited* and *bisection limited* routing problems, a distinction which is based on the number of packets each processor has to route. We concentrate on permutation problems on a ring of processors. The reason for doing so, is because, before trying to attack the problem for the more general case of r-dimensional meshes and tori, we must have a full understanding of the problem in its simplest form. We prove a new lower bound for distance limited problems on rings and we give an algorithm that matches the lower bound. For the case of bisection limited routing problems, we present an algorithm that

[1] Currently at Computer Science Department, Dartmouth College, Hanover, NH 03755

routes the packets in near optimal time.

A *ring of processors* is defined to be a graph $G = (V, E)$ where, $V = \{i | i = 0, 1, 2, ..., n-1\}$ and an edge $e = (i, j)$ belongs to E if $|j - i| = 1$ or $|j - i| = n - 1$. At any step, each processor can communicate with both of its neighbors.

We define the *distance along the shortest path* between processors $P_1 = i$ and $P_2 = j$, denoted $D_s(P_1, P_2)$, to be the minimum number of links that a packet has to traverse starting from processor P_1 and destined for processor P_2. Obviously, $D_s(P_1, P_2) = D_s(P_2, P_1)$. Formally, for processors $P_1 = i$ and $P_2 = j$, $j > i$, we define $D_s(P_1, P_2) = \min\{(j - i), n - (j - i)\}$.

In a *permutation routing problem* each processor has one packet to transmit to any other processor. At the end, each processor receives exactly one packet. In the *multi-packet permutation problem* each processor has k packets all of which are destined for the same processor. At the end, each processor receives exactly k packets. This problem arises when a single packet in the permutation routing problem consists of k flits. Some work has already been done on square meshes for this case: Simvonis and Makedon [8] treated the k flits as an unbreakable "snake", while Kunde and Tensi [4] routed the flits of a packet independently. Up to now, however, no work has been done on rings.

The remainder of the paper is organized into sections as follows. In Section 2, we define the classes of distance limited and bisection limited routing problems. In Section 3, we concentrate on distance limited problems on rings of n processors. We present a lower bound of $\frac{2n}{3}$ steps, and we give an algorithm that matches that bound. In Section 4, we investigate the bisection limited problem on a ring of n processors. The known lower bound for this problem is $\frac{kn}{4}$ routing steps. We present an algorithm that routes any problem in at most $\frac{kn}{4} + \frac{5n}{2}$ routing steps. Finally, in Section 5, we discuss further work that has to be done in this area.

2 Two Types of Routing Problems

We obtain lower bounds on the number of steps required to solve a routing problem using two different arguments. The first one is a lower bound based on the maximum distance a packet has to travel (*distance bound*). The second one is based on the *bisection bound* of the network used. Then, the lower bound is *max(distance bound, bisection bound).* For the case of r-dimensional meshes of side-length n, the distance bound is $r(n-1)$ and the bisection bound is $\frac{nk}{2}$, where k is the number of packets each processor holds. Thus, the lower bound on the number of steps required to solve the multipacket permutation routing problem on the r-dimensional mesh is $\max\{r(n-1), \frac{nk}{2}\}$. Similarly, for the torus, the lower bound is $\max\{\frac{r(n-1)}{2}, \frac{nk}{4}\}$.

It is clear from the above that we can divide the routing problems into two categories: the *bisection-limited problems* and the *distance-limited problems.* A problem is bisection-limited if the bisection lower bound is greater than the distance lower bound. Otherwise,

we say that the problem is distance-limited. For r-dimensional meshes and tori, we obtain that a problem is distance-limited if the number of packets per processor is $k \leq 2r$. Otherwise it is bisection-limited.

It should be pointed out that the division of the routing problems into the two categories is based on worst case scenarios. Thus, there are problems that, according to the above distinction, are bisection limited, ($k > 2$ for rings), and still can be solved in less time than that indicated by the distance limit. One such trivial example is when all processors have to route their packets to the processor immediately after them in the clockwise direction. Obviously, this routing problem can be solved in k steps.

In the rest of the paper we will concentrate on rings of processors. For a ring ($r = 1$) the problem is distance limited if $k \leq 2$ and bisection limited otherwise.

3 Distance Limited Routing Problem on a Ring

In what follows, we demonstrate a better lower bound for the case of distance limited routing problems on rings. The channel utilization of the network is taken into consideration in order to obtain the new lower bound. Then, we give an algorithm that matches the lower bound.

3.1 Lower Bound on a Ring of Processors

Let us assume that we have a ring of n processors and we want to route a multipacket permutation on it. Each processor has two packets that will be routed independently. Consider the following situation: Initially, any processor i contains 2 packets destined for processor $(i + \frac{n}{3})$ mod n. Processors are numbered from $0...n-1$ in the clockwise direction (CW) (Figure 1).

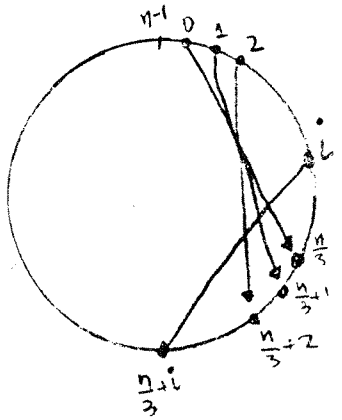


Figure 1. The instance of the permutation routing problem that requires $\frac{2n}{3}$ steps.

Hence, in the counter-clockwise direction (CCW), each packet has to travel distance $\frac{2n}{3}$. If some packet decides to move in the CCW direction, then at least $\frac{2n}{3}$ steps are required, since the distance between origin and destination is exactly $\frac{2n}{3}$ in the CCW direction. So, if we want to achieve a better routing time, we have to send all packets in the CW direction. In this case, each of a total of $2n$ packets will travel for $\frac{n}{3}$ steps. Then, the total movement (number of wire crossings) is $\frac{2n^2}{3}$. Since all packets are moving in the same direction, only n communication links are used. Hence, this movement requires at least $\frac{2n}{3}$ routing steps to be accomplished. Thus, we have:

Theorem 1. There is a distance limited routing problem on a ring of n processors that requires $\frac{2n}{3}$ routing steps for its solution.

3.2 An Algorithm that Tightens the Lower Bound

An algorithm that tightens the lower bound given in Theorem 1 is the following:

Algorithm *Route_on_a_Ring_1*

At step 1 Each processor determines the minimum distance its packets have to travel, say s, where $0 \leq s \leq \frac{n}{2}$. It also determines the direction in which the packets have to travel so that their distance to the destination is s. Let this direction be denoted by K (CW or CCW).

- If $s \leq \frac{n}{3}$, then both packets are send in direction K.
- If $\frac{n}{3} < s \leq \frac{n}{2}$, then the processor sends one packet in the CW direction and one in the CCW direction.

at step $i, i > 1$ Each processor transmits a packet toward its destination, if it has one.

A processor never changes the direction of a packet.

The packet that has to go further in a given direction has higher priority.

Lemma 1. At any time t, using Algorithm *Route_on_a_Ring_1*, a processor has at most 2 packets that want to move in the same direction.

Proof. The Lemma is obvious, since: i) each processor transmits a packet toward a given direction, if it has one, ii) at any step, it can receive at most one packet that must be sent in that direction, and, iii) the initial load of each processor is at most 2 packets that want to travel in the same direction. ∎

Lemma 2. Using Algorithm *Route_on_a_Ring_1*, there are at most $\frac{2n}{3}$ packets that want to cross any "cut" in the same direction.

Proof. W.l.o.g., we assume a "cut" after processor $n-1$, and we will examine packets moving only in the CW direction. The proof for the CCW movement is symmetric. Let processors $P_{\frac{n}{3}}, ..., P_{\frac{2n}{3}-1}$ constitute segment A of the ring, and processors $P_{\frac{2n}{3}}, ..., P_{n-1}$ constitute segment B of the ring (Figure 2).

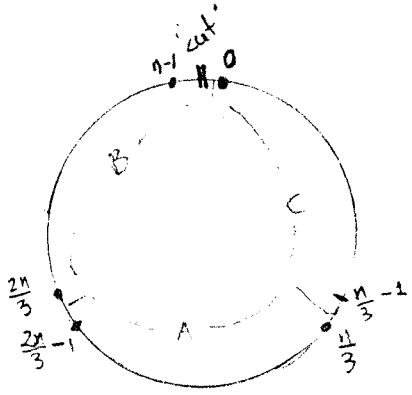


Figure 2. The ring is divided into three sections that are used in the proof of Lemma 2.

Initially, each processor in segment A has at most 1 packet that wants to move in the CW direction and also wants to cross the "cut". Assume that the number of packets in segment A that want to cross the "cut" is m , $0 \leq m \leq \frac{n}{3}$. Then, these packets must be destined for segment C, where processors $P_0, ..., P_{\frac{n}{3}-1}$ constitute segment C. Since we are examining permutation routing, for any packet in segment A that wants to cross the "cut" in the CW direction, there must exist a unique destination in segment C. This means that, there exist m positions in segment C, none of which can be the destination of a packet that is initially in segment B. Thus, in segment B, there must exist m processors that initially have no packets that want to cross the "cut" and are also a distance $d \leq \frac{n}{3}$ from the "cut". (Note that these processors might have only one packet that wants to cross the "cut"). So, the total number of packets which are initially in segment B and want to cross the "cut" is at most $(\frac{2n}{3} - k)$. This implies that the total number of packets that want to cross the "cut" in CW direction is at most $(\frac{2n}{3} - m) + m = \frac{2n}{3}$. ∎

Theorem 2. Using Algorithm *Route_on_a_Ring_1*, a distance limited permutation problem on a ring of processors can be solved in $\frac{2n}{3}$ steps.

Proof. In order to prove this, we need to show that all packets that want to cross any "cut" in a given direction, will have done so after $\frac{2n}{3}$ steps. W.l.o.g, let us assume the

"cut" is after processor $\frac{2n}{3}$ in the CW direction. We simulate the routing process as follows: Assume we have two tapes, tape A and B. Tape A has $\frac{2n}{3}$ cells and can move to the right. Tape B has $\frac{n}{3}$ cells, is not allowed to move, and is placed, initially, on top of the $\frac{n}{3}$ rightmost cells of tape A. Each cell of a tape can hold at most one "pebble". A pebble represents a packet that wants to cross the "cut" in the CW direction. If a pebble is placed on a cell of tape B, then a pebble must also exist on the underlying cell of tape A. We now observe that, initially, we have the following situation: If k pebbles are on tape B, then at least k cells in the $\frac{n}{3}$ leftmost positions of tape A are empty. We associate the i^{th} pebble of tape B, $0 \leq i < \frac{2n}{3}$, where we count from left to right, with the i^{th} empty cell ("hole") of tape A, where we count from right to left (Figure 3). This is a 1-1 relation.

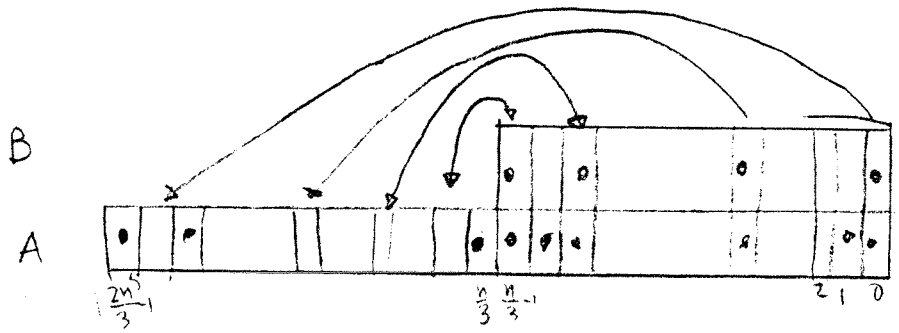


Figure 3. The simulation of the routing by two tapes as used in the proof of Theorem 2.

Now, we start moving tape A to the right. In the worst case, each pebble in tape B will drop into its corresponding "hole" in tape A. Since the whole tape A will cross the "cut" after exactly $\frac{2n}{3}$ steps, all pebbles cross the "cut" in at most $\frac{2n}{3}$ steps. Thus, all packets that want to cross the "cut" can do so in $\frac{2n}{3}$ steps. ∎

4 Bisection Limited Routing Problem on a Ring

In this section, we consider bisection limited problems on rings of n processors. Recall, from Section 2, that a problem is bisection limited for rings when k, the number of packets per processor, is greater than 2. The lower bound based on the bisection of the ring is $\frac{kn}{4}$. We give an algorithm that completes the routing in $\frac{kn}{4} + \frac{5n}{2}$ steps. The trivial greedy algorithm that routes each packet along the shortest path to its destination takes, in the worst case $\frac{kn}{2}$ steps.

4.1 The Algorithm

Before we proceed with the description of the algorithm, we need to give some definitions for certain variables that we use: Let S_i denote the distance that the packets initially

located at processor P_i have to travel along the shortest path to their destination. S_i can be written as $S_i = \lambda_i n$, $0 \leq \lambda_i \leq \frac{1}{2}$, where λ_i is a coefficient used in our algorithm. Our algorithm routes a fraction of the packets which are initially at processor P_i along the shortest path to their destination, and routes the remaining packets along the longest path. The number of packets that are routed along the shortest path and are originated at processor P_i is denoted by σ_i.

Algorithm *Route_on_a_Ring_2*

At step 1 Each processor P_i determines the number of packets σ_i it will send along the shortest path using the following rule:

$\sigma_i = k - \lfloor \lambda_i k \rfloor$

It then adds σ_i of the k packets to a queue associated with the link on the shortest path, and the remaining packets to the queue associated with the other link.

The packets at the front of the queues are transmitted.

at step $i, i > 1$ Each processor transmits a packet toward its destination, if it has one.

A processor never changes the direction of a packet.

The packets are transmitted using a FIFO policy .

Our efforts to analyze Algorithm *Route_on_a_Ring_2* in a way similar to the analysis of Algorithm *Route_on_a_Ring_1* were not successful. In particular, we were not able to introduce in a proof of that kind the fact that the routing problem is a permutation. So, we proceed with a totally different approach. Again, we consider the number of packets that cross any "cut" in any direction. But now, we prove that the given routing problem is "easier" to be solved than a special kind of routing problem for which we can make statements regarding its complexity. In what follows we assume that the ring consists of an even number of processors.

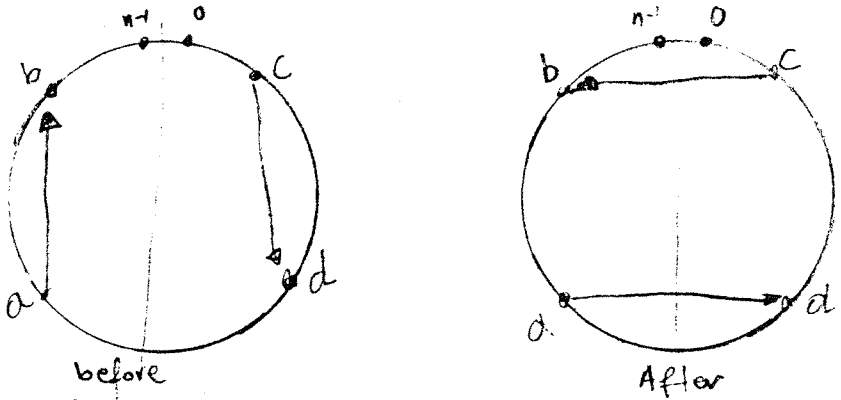
Definition Let a ring be divided by a diameter into two sections. Each section consists of $\frac{n}{2}$ processors. A multipacket routing problem in which all packets from one section are destined for the other is called a *symmetric multipacket routing problem.* All other multipacket routing problems are *asymmetric.*

Lemma 3 Assume any "cut" in any direction. Using Algorithm *Route_on_a_Ring_2*, an asymmetric multipacket routing problem with initial load of k packets at each processor sends at most as many packets to cross the "cut" in the given direction as a symmetric one with load of $k+1$ packets per processor where, the extra packet is routed towards the "cut".

Proof. Without loss of generality, assume a ring of n processors, and a "cut" between processors P_0 and P_{n-1}. We will concentrate in the number of packets that cross the "cut" in the CW direction. A diameter that passes through the "cut" divides the ring into two sections. Section A consists from processors $P_0, P_1, ..., P_{\frac{n}{2}-1}$ and Section B consists of processors $P_{\frac{n}{2}}, P_{\frac{n}{2}+1}, ..., P_{n-1}$. We will show how to transform any asymmetric multipacket routing problem with initial load of k packets to a symmetric problem with initial load of $k+1$ packets. Furthermore , the solution of the new routing problem will require greater or equal number of routing steps. (Both problems are solved by Algorithm *Route_on_a_Ring_2*.) First observe that if there is a processor in section A that has packets destined for section A, then, there exist a processor in section B that has packets destined for section B. The above observation follows from the pigeonhole principle.

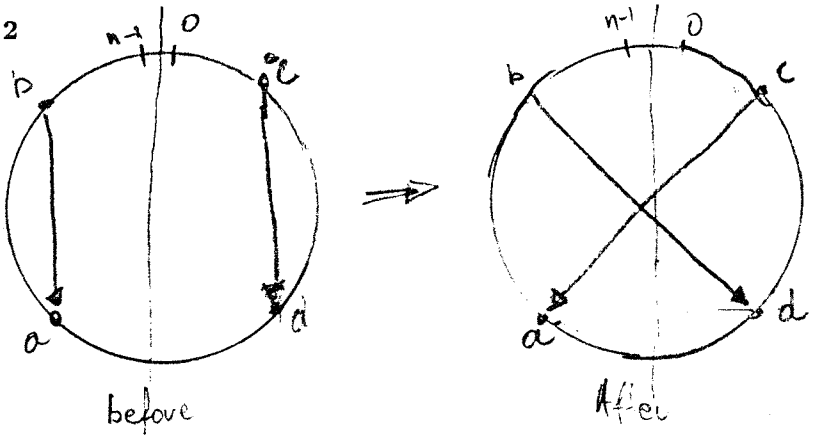
Our transformation consists by picking two processors, one in each section, that have packets destined for the sections they belong. Then, the destinations will be switched. By performing the above transformation for at most $\frac{n}{2}$ times, we will get a symmetric multipacket routing problem. Now it remains to prove that if we load each processor at the new problem with one additional packet, the new problem is at least as hard as the initial one. We distinguish four cases.

Case 1



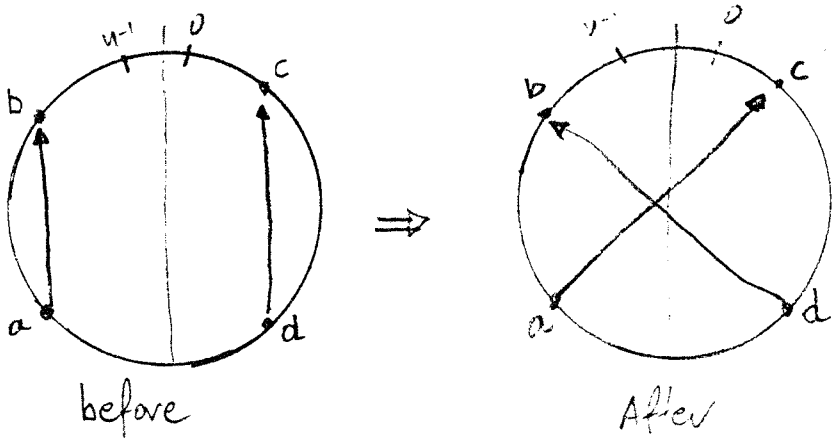
The packets at processor a are destined for processor b. Also, the packets at processor c are destined for processor d. After the switch of the destinations, the packets at processor a are destined for processor d and the packets at processor c are destined for processor b. Observe that before the switch no packet wants to cross the "cut". After the switch a portion of the packets located at processor a might cross the "cut" in the CW direction. So, the new problem is of greater or equal difficulty with the original one.

Case 2



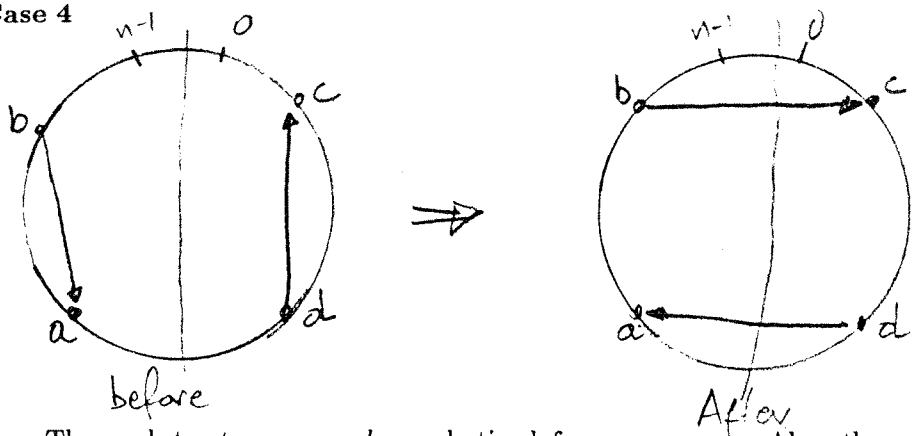
The packets at processor b are destined for processor a. Also, the packets at processor c are destined for processor d. After the switch of the destinations, the packets at processor b are destined for processor d and the packets at processor c are destined for processor a. Observe that before and after the switch only packets that are initially located at processor b will cross the "cut" in the CW direction. After the switch, at least the same number of packets will cross the "cut" in the CW direction since the distance the packets have to travel in the CW direction is reduced.

Case 3



The packets at processor a are destined for processor b. Also, the packets at processor d are destined for processor c. After the switch of the destinations, the packets at processor a are destined for processor c and the packets at processor d are destined for processor b. Before the switch only packets initially located at processor d will cross the "cut" in the CW direction. After the switch only packets located at processor a will cross the "cut" in the CW direction. But the number of packets that cross the "cut" now, is at least as large as before. This is because the distance the packet which cross the "cut" have to travel is reduced.

Case 4



The packets at processor b are destined for processor a. Also, the packets at processor d are destined for processor c. After the switch of the destinations, the packets at processor b are destined for processor c and the packets at processor d are destined for processor a. This is the most interesting case. Before the switch packets from both origins want to cross the "cut" while, after the switch, only packets initially located at processor d want to do so. Before the switch, processor b sends at most $\frac{k}{n}(ab)_{cw}$ packets towards the "cut". Processor d sends at most $\frac{k}{n}(cd)_{cw}$ packets towards the "cut". Thus, before the switch, at most $\frac{k}{n}((ab)_{cw} + (cd)_{cw})$ packets cross the "cut" in the CW direction. After the switch at least $\frac{k}{n}(cb)_{cw} - 1$ packets will cross the "cut" in the CW direction. But $(cb)_{cw} \geq (ab)_{cw} + (cd)_{cw}$. Thus, if we load processor b with one extra packet and we force it to cross the "cut" in the CW direction, the new problem is of equal or greater difficulty.

The observation that all of the transformations needed to convert an asymmetric multipacket problem into a symmetric one might be of that described in case 4 proves the lemma. ∎

Lemma 4 Assume a symmetric multipacket routing problem with initial load of k packets per processor around a given "cut". Also assume that its solution by Algorithm *Route_on_a_Ring_2* causes X to cross the "cut" in a given direction. Then, the symmetric multipacket routing problem where the packets at any processor are destined after $\frac{n}{2}$ positions in the given direction, causes at least $X - \frac{3n}{2}$ packets to cross the "cut" in the direction under consideration. (ie load each pro with 3 extra packs
Proof Omitted.

Lemma 5 Using Algorithm *Route_on_a_Ring_2*, there are at most $\frac{kn}{4} + \frac{5n}{2}$ packets that want to cross any "cut" in the same direction.

Proof Lemmata 3 and 4 imply that a routing problem that is symmetric, and has initial load of $k + 4$ packets per processor destined for the processor located after $\frac{n}{2}$ positions in the CW direction, is harder than any other multipacket routing problem with initial load k packets per processor. (The 4 extra packets will be routed in the CW direction.)

Algorithm *Route_on_a_Ring_2* will send at most $\frac{kn}{4} + \frac{n}{2}$ packets to cross the "cut". The routing of the 4 extra packets per processor will contribute $2n$ more packets to that number. ∎

Theorem 3 Using Algorithm *Route_on_a_Ring_2*, a bisection limited permutation problem on a ring of processors can be solved in $\frac{kn}{4} + \frac{5n}{2}$ routing steps.

Proof. The theorem is implied from Lemma 5. If at every step of the algorithm one packet wants to cross the "cut", the theorem is obviously true. In the case where there is a period during the execution of the routing algorithm that no packet crosses the "cut", say μ steps, then there are μ processors that do not have any packets that want to cross the "cut". Thus, the number claimed at Lemma 5 was overestimated by at least μ. So, the theorem holds. ∎

5 Conclusions - Further work

In this paper, we have examined multipacket routing problems on rings. We divided these problems into two categories, distance limited and bisection limited routing problems. We presented a new lower bound for the case of distance limited problems and we gave an algorithm that tightens the lower bound. For the case of bisection limited problems, we presented an algorithm that solves any problem within $\frac{kn}{4} + \frac{5n}{2}$ routing steps. We believe that the number of routing steps that Algorithm *Route_on_a_Ring_2* actually requires is at most $\frac{kn}{4} + n$, and we are working on improving our proofs. No matter if our conjecture is true, we have succeed to present an algorithm that approximates within an additive factor the number of routing steps required in the worst case.

References

[1] B. Aielo, F.T. Leighton, B. Maggs, M. Newman, "Fast Algorithms for Bit-Serial Routing on a Hypercube", Proceedings of the 2^{nd} Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '90, July 2-6, 1990, Crete, Greece.

[2] D. Krizanc, S. Rajasekaran, Th. Tsantilas, "Optimal Routing Algorithms for Mesh-Connected Processor Arrays", VLSI Algorithms and Architectures (AWOC'88), J. Reif, editor, Lecture Notes in Computer Science 319, 1988, pp. 411-422.

[3] M. Kunde, "Routing and Sorting on Mesh-Connected Arrays", VLSI Algorithms and Architectures (AWOC'88), J. Reif, editor, Lecture Notes in Computer Science 319, 1988, pp. 423-433.

[4] M. Kunde,T. Tensi, "Multi-Packet Routing on Mesh Connected Arrays", Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA'89, June 1989, pp. 336-343.

[5] F.T. Leighton, "Average Case Analysis of Greedy Routing Algorithms on Arrays", Proceedings of the 2^{nd} Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '90, July 2-6, 1990, Crete, Greece.

[6] F.T. Leighton, F. Makedon, I.G. Tollis, "A 2n-2 Algorithm for Routing in an $n \times n$ Array With Constant Size Queues", Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA'89, June 1989, pp. 328-335.

[7] F. Makedon, A. Simvonis, "Fast Parallel Communication on Mesh Connected Machines with Low Buffer Requirements", Proceedings of the 1990 IEEE International Conference on Computer Design (ICCD '90).

[8] F. Makedon, A. Simvonis, "On Bit-Serial Packet Routing for the Mesh and the Torus", Proceedings of the 3^{rd} Symposium on the Frontiers of Massively Parallel Computation, October 8-10 1990, pp. 294-302.

[9] J.Y. Ngai,C.L. Seitz, "A Framework for Adaptive Routing in Multicomputer Networks", Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA'89, June 1989, pp. 1-9.

[10] S. Rajasekaran, R. Overholt, "Constant Queue Routing on a Mesh", to appear in the Journal of Parallel and Distributed Computing.

[11] A.G. Ranade, "How to Emulate Shared Memory", Proceedings of the 28^{th} IEEE Symposium on Foundation of Computer Science, 1987, pp. 185-194.

[12] L.G. Valiant, G.J. Brebner, "Universal Schemes for Parallel Communication", Proceedings of the 13^{th} Annual ACM Symposium on the Theory of Computing, May 1981, pp. 263-277.